SUMMARY OF ANSI X9.63
Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography

1. Scope

ANSI X9.63 defines key establishment schemes that employ asymmetric cryptographic techniques. The arithmetic operations involved in the operation of the schemes take place in the algebraic structure of an elliptic curve over a finite field. Both key agreement and key transport schemes are specified. The schemes may be used by two parties to compute shared keying data that may then be used by symmetric schemes to provide cryptographic services, e.g. data confidentiality and data integrity.

2. Contents

The asymmetric key establishment schemes in ANSI X9.63 are used by an entity *U* who wishes to establish a symmetric key with another entity *V*. Each entity has at least one elliptic curve key pair. If *U* and *V* simultaneously execute a scheme with corresponding keying material as input, then at the end of the execution of the scheme, *U* and *V* will share keying data. The keying data can then be used to supply keys for symmetric algorithms.

ANSI X9.63 specifies a variety of asymmetric key establishment schemes. Eleven key agreement schemes and two key transport schemes are provided in the document. A variety of schemes are specified because of the wide variety of services that it may or may not be desirable for a key establishment scheme to provide, depending on the environment in which the scheme is going to be used.

The standard also contains specifications for:

- Elliptic curve domain parameter generation and validation
- Key pair generation and public key validation
- Challenge generation
- Diffie_Hellman primitive
- MQV primitive
- Cryptogrpahic hash
- Key derivation function
- MAC calculation
- Encryption transformation
- Signature scheme.

In additon, for each scheme specified in the standard, an assessment of the security that is afforded by the scheme is provided.

3. Crypographic Ingredients

3.1 Elliptic Curve Keying Material

The keying material needed to perform key establishment or key transport as specified by this standard includes a set of domain parameters and one or more elliptic curve key pairs.

The domain parameters include:

- q the number of elements in the field
- a,b elements of Fq that define an elliptic curve E over F_q
- G a distinguished point on an elliptic curve that is the base point or generating point
- n the order of the base point G
- h the co-factor (the number of points on the curve, divided by n)

The domain parameters may be distributed, for example, in a public key certificate. Two classes of domain parameters may exist: static domain parameters or ephemeral domain parameters. Static domain parameters may be used with either static or ephemeral keys. Static domain parameters are designated with a subscript of s when ephemeral domain parameters are also involved in the key establishment scheme (i.e., q_s, a_s, b_s, G_s, n_s, h_s). Ephemeral domain parameters are used with ephemeral keys and are designated with a subscript of e (i.e., q_e, a_e, b_e, G_e, n_e, h_e).

An elliptic curve key pair consists of a private key d, and a public key Q. Two classes of key pairs may be used in a scheme: static key pairs (d_s, Q_s) and ephemeral key pairs (d_e, Q_e). Static key pairs are longer-lived; the public key from a static key pair (Q_s) may be included in a public key certificate. Ephemeral keys are shorter lived (e.g., for the duration of a message or a communication session).

3.2 Domain Parameter Generation and Validation

ANSI X9.63 provides primitives for both domain parameter generation and validation. The primitives differ depending on the characteristics of the underlying field, i.e., for the fields F_p and $F_2{}^m$. Domain parameter validation may be used to verify that the domain parameters satisfy certain basic criteria (e.g., $q=p$ is an odd prime; a, b, x_G, y_G are integers in the interval [0, p-1]; etc.). Domain parameter validation could be performed by a Certificate Authority (CA) or some other trusted party.

3.3 Key Pair Generation and Public Key Validation

ANSI X9.63 provides primitives for key pair generation and public key validation. Both static keys and ephemeral keys may be generated using the same primitives. Public key validation may be used to verify that an entity's public key Q appears to satisfy certain basic criteria (e.g., Q is not the point at infinity; x_Q and y_Q are elements in the field indicated by the domain parameters; etc.). A static public key could be validated, for example, by a CA, a trusted party or a responder in a key establishment protocol. An ephemeral public key might be validated by the responder in a key establishment protocol.

3.4 Diffie-Hellman Primitive (dhp)

The Diffie-Hellman primitive is used to compute a shared secret value. Two variants of the Diffie-Hellman primitive are provided – one without the use of the co-factor h, the other using the co-factor.

Input:

- a private key d_A (A is the local party)
- a public key Q_B (B is the other party)

Compute:

$P = [h]d_A Q_B$ (where P is a point and h is the co-factor)

$z = x_p$ (where x_p is the x coordinate of P)

Output: z, the shared value

3.5 MQV Primitive

The MQV primitive derives a shared secret value from two secret keys owned by the local party (A) and two public keys owned by the other party (B). Note that in the schemes, the key pairs defined below may be set to the same values.

Input:

- 2 key pairs for A – $(d_{1,A}, Q_{1,A})$ and $(d_{2,A}, Q_{2,A})$
- 2 public keys for B –$Q_{1,B}$ and $Q_{2,B}$

Compute:

$implicitsig_{\mathrm{U}} = d_{2,U} + (avf(Q_{2,U}) \times d_{1,U})$ (mod n)

$P = h \times implicitsig_U \times (Q_{2,V} + (avf(Q_{2,V}) \times Q_{1,V}))$

$z = x_p$

Output: z, the shared value.

3.6 Associate Value Function

The associate value function will be used to compute an integer associated with an elliptic curve point. The integer is approximately half the length of the field elements associated with the elliptic curve point.

Input:

- A valid set of domain parameters
- A point P ≠ a point at infinity

Compute:

$x_P' = x_P \pmod{2^{\lceil f/2 \rceil}}$ (where $f = \lceil \log_2 n \rceil$)

$avf(P) = x_P' + 2^{\lceil f/2 \rceil}$

Output: *avf(P)*

3.7 Cryptographic Hash Function

The crypographic hash functions are used to calculate the hash value associated with a bit string. The functions are used by the key derivation function specified in Section 3.8. Any ANSI-approved hash function that offers 80 bits of security or more may be used, i.e. any ANSI-approved hash function whose output is 160 bits or more. Possibilities include the hash function SHA-1 specified in ANSI X9.30, Part 2 (*The Secure Hash Algorithm* 1).

Input: A bit string *Data*

Compute: $Hash = H(Data)$ (where H is the established hash function)

Output: *Hash*

3.8 Key Derivation Function (kdf)

The key derivation function is used by the key agreement schemes to compute keying data from a shared secret value. The key derivation function will also be used by the asymmetric encryption schemes. The key derivation function is a simple construction based on a hash function.

Input:

- Shared value z
- Key length *keydatalen*
- Optional *SharedInfo*

Compute:

Set $counter = 1$

For $i = 1$ to $j = \lceil keydatalen/hashlen \rceil$, do:

$Hash_i = H(Z \,||\, counter \,||\, [SharedInfo])$

Increment *counter*

Increment i

Let $HHash_j$ denote $Hash_j$ if *keydatalen*/*hashlen* is an integer, and let it denote the ($keydatalen - (hashlen \times j)$) leftmost bits of $Hash_j$ otherwise

Set $KeyData = Hash_1 || Hash_2 || \ldots || Hash_{j-1} || HHash_j$

Output: *KeyData*

3.9 MAC Scheme

A tagging transformation and tag checking transformation is associated with the message authentication code (MAC) scheme. The MAC scheme is used by some key agreement schemes to provide key confirmation, and by the augmented encryption scheme in Section XXXX???. Any ANSI-approved MAC that offers 80 bits of security or more may be used, i.e. any ANSI-approved MAC that uses keys of length 80 bits or more and that outputs tags of length 80 bits or more. Possibilities, therefore, include HMAC specified in ANSI X9.71, *Keyed Hash Message Authentication Code*.

Input:

- A bit string *Data* to be MACed
- A bit string *MacKey*

Compute:

$MacTag = MAC_{MacKey}(Data)$

3.10 Asymmetric Encryption Scheme

The asymmetric encryption scheme is used as follows. The sender uses the encryption transformation of the scheme to encrypt some data. The recipient, after being sent the encrypted data, decrypts the encrypted data using the decryption transformation of the scheme. The asymmetric encryption scheme is used by the key transport schemes.

3.10.1 Encryption Transformation

Input:

- *Data* to be encrypted of length *datalen* bits
- Public key of the recipient Q_B
- Two optional bit strings $SharedData_1$ and $SharedData_2$

Compute:

Generate (d_e, Q_e)
$z = dhp(d_e, Q_B)$
$EncKey \| MacKey = kdf(z, datalen+mackeylen, [SharedData_1])$
$MaskedEncData = Data \oplus EncKey$
Compute *MacTag* for $MacData = MaskedEncData \| [SharedData_2]$

Output: $Q_e \| MaskedEncData \| MacTag$

3.10.2 Decryption Transformation

Input:
- $EncryptedData = Q_{e'} \mid\mid MaskedEncData' \mid\mid MacTag'$
- Private key d_B
- Two optional bit strings $SharedData_1$ and $SharedData_2$

Compute:

Validate Q_e'
$z = dhp(d_B, Q_e')$
$EncKey \mid\mid MacKey = kdf(z, maskedencdatalen+mackeylen, [SharedData_1])$
$EncData = MaskedEncData'$ Å $EncKey$
Verify that *MacTag*' is correct for *MaskedEncData* || [$SharedData_2$] using *MacKey*

Output: *Data*

3.11 Signature Scheme

The signature scheme is used as follows. The sender uses the signing transformation to compute a signature on some data. The recipient, after being sent the data and signature, checks the validity of the signature using the verifying transformation. The signature scheme is used by the 3-pass key transport scheme specified in Section XXX. The signature scheme supported is the Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA is specified in ANSI X9.62, *The Elliptic Curve Digital Signature Algorithm (ECDSA).* The signature process produces two integers: *rsig* and *ssig*.

4. Security Attributes

A number of security attributes may be provided by a particular key agreement or key transport scheme. These attributes are:
- Implicit Key Authentication (IKA) – establishes the identity of the other party; provided by a static key that is bound to the other party's identity.
- Explicit Key Authentication (EKA) – knowledge that the other party did indeed calculate the shared key; may be provided by doing key confirmation in a scheme that provides implicit key authentication.
- Forward Secrecy (FS) – the assurance provided to an entity that the session key established with another entity will not be compromised by the compromise of either entity's static private key in the future; provided by the use of an ephemeral public key.
- Entity Authentication (EA) – consists of identification and "liveness", whereas EKA is concerned with identification and key possession. If explicit confirmation is deemed adequate to demonstrate liveness, then it is possible for schemes that achieve EKA to also support EA. Whether or not the two are synonymous depends upon the interpretations of "entity" and "liveness", and on the timespan for the protocol that realizes the key agreement scheme. Therefore, the EA attribute does not follow automatically from EKA. ANSI X9.42

does not address entity authentication more specifically because EKA is neither always necessary nor always sufficient to achieve EA.

- Known-Key Security (K-KS) – assurance that a particular session key will not be compromised as a result of the compromise of other keys; provided by using an ephemeral key to compute the key(s) for only one session.
- Unknown Key-share Resilience (U-KS) – assurance provided to one party (A) that if party A and party B share a session key, then party B does not mistakenly believe the session key is shared with an entity other than party A.
  Key-Compromise Impersonation Resilience (K-CI) – assurance provided to a party (A) that, even if an adversary somehow obtains party A's static private key, the adversary cannot successfully impersonate another party to party A.

5. Key Agreement Schemes

In the following schemes, the initiator is party U, and the responder is party V. The acquisition of static public keys is outside the scope of ANSI X9.63, but is required for the key establishment process. These acquisitions are shown as "dashed arrows".

5.1 Ephemeral Unified Model

Ephemeral parameters (q_e, a_e, b_e, G_e, n_e, and h_e) are generated for the "system" (a community of users). The hash function must be agreed upon.

This scheme corresponds to the dhEphem scheme in ANSI X9.42. The scheme provides Known-key Security if explicit authentication of all session keys is supplied for all session keys. Interactive communications are required, i.e., both parties must actively participate in the key agreement process.

Party U	Transmissions	Party V
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q_e, a_e, b_e, G_e, n_e, and h_e)	$\xrightarrow{Q_{e,U}}$	
		Validate $Q'_{e,U}$
	$\xleftarrow{Q_{e,V}}$	Generate ephemeral key pair ($d_{e,V}$, $Q_{e,V}$) using (q_e, a_e, b_e, G_e, n_e, and h_e)
Validate $Q'_{e,V}$		
Compute the shared value ($Z_e = dhp(d_{e,U}, Q'_{e,V})$) using ($q_e$, a_e, b_e, G_e, n_e, and h_e): $Z_e = [h]d_{e,U}Q_{e,V}$		Compute the shared value ($Z_e = dhp(d_{e,V}, Q'_{e,U})$) using ($q_e$, a_e, b_e, G_e, n_e, and h_e): $Z_e = [h]d_{e,V}Q_{e,U}$
Determine the shared key: $KeyData = kdf(Z_e, keydatalen, [SharedData]$		Determine the shared key: $KeyData = kdf(Z_e, keydatalen, [SharedData]$

5.2 One Pass Diffie Hellman

Static parameters (q, a, b, G, n, and h) are generated for the "system" (a community of users). Each entity that acts as a responder generates a static key pair (d_s, Q_s) using the static parameters. The entities must agree upon a hash function.

This scheme corresponds to the dhOneFlow scheme of ANSI X9.42. The scheme provides Implicit Key Authentication and Key Compromise Impersonation resilience to the initiator. Interactive or store-and-forward communications (only one party actively participates) are required.

Party U	Transmissions	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q, a, b, G, n, and h)	$Q_{e,U}$ ————————————>	
		Validate $Q'_{e,U}$
Compute the shared value ($Z = dhp(d_{e,U}, Q'_{s,V})$) using ($q$, a, b, G, n, and h): $Z=[h]d_{e,U}Q_{s,V}$		Compute the shared value ($Z = dhp(d_{s,V}, Q'_{e,U})$) using ($q$, a, b, G, n, and h): $Z=[h]d_{s,V}Q_{e,U}$
Determine the shared key: *KeyData = kdf(Z, keydatalen,* [*SharedData*]		Determine the shared key: *KeyData = kdf(Z, keydatalen,* [*SharedData*]

5.3 Static Unified Model

Static parameters (q_s, a_s, b_s, G_s, n_s, and h_s) are generated for the "system" (a community of users). Each entity generates a static key pair (d_s, Q_s) using the static parameters. The hash function must be agreed upon.

This scheme corresponds to the dhstatic scheme of ANSI X9.42. The scheme provides Implicit Key Authentication to both parties, and Unknown-Keyshare Resilience to both parties if knowledge of the private key (also known as proof of possession) was checked during the certification of the static public keys. Applications involving interactive or store-and-forward communications or neither party online can be used.

Party U	Transmission	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Compute the shared value (Z_s = $dhp(d_{s,U}, Q_{s,V})$) using (q_s, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,U}Q_{s,V}$		
Determine the shared key: $KeyData = kdf(Z_s,$ $keydatalen$, [$SharedData$]	Notify V that U wants to communicate ————————————>	
	$Q_{s,U}$ – – – – – – – – – – – – –>	Acquire U's static public key ($Q_{s,U}$)
		Compute the shared value (Z_s = $dhp(d_{s,V}, Q_{s,U})$) using (q_s, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,V}Q_{s,U}$
		Determine the shared key: $KeyData = kdf(Z_s,$ $keydatalen$, [$SharedData$]

5.4 Combined Unified Model with Key Confirmation

This scheme is a hybrid of the Ephemeral Unified Model and the Static Unified Model schemes. A MAC is used to provide Key Confirmation. This scheme provides an example of how key confirmation could be provided.

Static and ephemeral parameters ((q_s, a_s, b_s, G_s, n_s, and h_s) and (q_e, a_e, b_e, G_e,n_e, and h_e)) are generated for the "system" (a community of users). Each entity generates a static key pair (d_s, Q_s) using the static parameters.. The entities must determine the MAC scheme and hash function to be used.

There is no corresponding scheme in ANSI X9.42. This scheme provides all security attributes except Key Compromise Impersonation resilience to both parties. Interactive communications are required.

Party U	Transmission	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q_e, a_e, b_e, G_e,n_e, and h_e)	$Q_{e,U}$ ————————————>	
		Validate $Q'_{e,U}$
	$Q_{s,U}$ – – – – – – – – – – – – – – >	Acquire U's static public key ($Q_{s,U}$)

		Generate ephemeral key pair ($d_{e,V}$, $Q_{e,V}$) using (q_e, a_e, b_e, G_e, n_e, and h_e)
		Compute the shared value ($Z_s = dhp(d_{s,V}, Q_{s,U})$)using ($q_s$, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,V}Q_{s,U}$
		Determine $MacKey = kdf(Zs, mackeylen, [SharedData_1]$
		Create $MacData_1 = 02 \| V \| U \| Q_{e,V} \| Q'_{e,U} \| [Text_1]$
	$Q_{e,V}$, [$Text_1$], $MacTag_1$ $\longleftarrow$	Create $MacTag_1 = Mac_{MacKey}(MacData_1)$
Validate $Q'_{e,V}$		
Compute the shared value ($Z_s = dhp(d_{s,U}, Q_{s,V})$) using ($q_s$, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,U}Q_{s,V}$		
Determine $MacKey = kdf(Z_s, mackeylen, [SharedData]$		
Create $MacData_1 = 02 \| V \| U \| Q'_{e,V} \| Q_{e,U} \| [Text_1]$		
Verify that the received $MacTag_1$ is correct for $MacData_1$		
Create $MacData_2 = 03 \| U \| V \| Q_{e,U} \| Q'_{e,V} \| [Text_2]$		
Create $MacTag_2 = Mac_{MacKey}(MacData_2)$	[$Text_2$], $MacTag_2$ $\longrightarrow$	
		Create $MacData_2 = 03 \| U \| V \| Q'_{e,U} \| Q_{e,V} \| [Text_2]$
		Verify that the received $MacTag_2$ is correct for $MacData_2$
Compute the shared value ($Z_e = dhp(d_{e,U}, Q'_{e,V})$)using ($q_e$, a_e, b_e, G_e,n_e, and h_e): $Z_e=[h]d_{e,U}Q_{e,V}$		Compute the shared value ($Z_e = dhp(d_{e,V}, Q'_{e,U})$) using ($q_e$, a_e, b_e, G_e,n_e, and h_e): $Z_e=[h]d_{e,V}Q_{e,U}$
Determine the shared key: $KeyData = kdf(Z_e, keydatalen, [SharedData_2]$		

5.5 One Pass Unified Model

The system parameters (q, a, b, G, n, and h) are generated for the “system” (a community of users). Each entity generates a static key pair (d_s, Q_s) using the system parameters. A hash function must be agreed upon.

This scheme corresponds to the dhHybridOneFlow scheme of ANSI X9.42. The scheme provides Implicit Key Authentication to both parties, Key-Compromise Impersonation resilience to the initiator, and Unknown Key-Share Resilience to both parties when knowledge of the private key is checked during the certification of the static public keys. Interactive or store-and-forward communications are required.

Party U	**Transmission**	**Party V**
Acquire V’s static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q, a, b, G, n, and h)	$Q_{e,U}$ ——————————————>	
		Validate $Q'_{e,U}$
	$Q_{s,U}$ – – – – – – – – – – – – – – >	Acquire U’s static public key ($Q_{s,U}$)
Compute the shared value ($Z_e = dhp(d_{e,U}, Q_{s,V})$) using ($q$, a, b, G, n, and h): $Z_e=[h]d_{e,U}Q_{s,V}$		Compute the shared value ($Z_e = dhp(d_{s,V}, Q'_{e,U})$) using ($q$, a, b, G, n, and h): $Z_e=[h]d_{e,V}Q'_{e,U}$
Compute the shared value ($Z_s = dhp(d_{s,U}, Q_{s,V})$) using ($q$, a, b, G, n, and h): $Z_s=[h]d_{s,U}Q_{s,V}$		Compute the shared value ($Z_s = dhp(d_{s,V}, Q_{s,U})$) using ($q$, a, b, G, n, and h): $Z_s=[h]d_{s,V}Q_{s,U}$
$Z = Z_e \parallel Z_s$		$Z = Z_e \parallel Z_s$
Determine the shared key: *KeyData = kdf*(*Z*, *keydatalen*, [*SharedData*]		Determine the shared key: *KeyData = kdf*(*Z*, *keydatalen*, [*SharedData*]

5.6 Full Unified Model

Static and ephemeral parameters ((q_s, a_s, b_s, G_s, n_s, and h_s) and (q_e, a_e, b_e, G_e,n_e, and h_e)) are generated for the “system” (a community of users). Each entity generates a static key pair (d_s, Q_s) using the static parameters. The hash function must be agreed upon.

This scheme corresponds to the dhHybrid2 scheme of ANSI X9.42. The scheme provides Implicit Key Agreement to both parties, Known-Key Security and Forward Secrecy to both parties if explicit authentication of all session keys is performed, and Unknown Keyshare Resilience to both parties if knowledge of the private key was checked during the certification of the static public keys. Interactive communications are required.

Party U	Transmission	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}, Q_{e,U}$)	$Q_{e,U}$ ————————————>	
		Validate $Q'_{e,U}$
	$Q_{s,U}$ – – – – – – – – – – – – – –>	Acquire U's static public key ($Q_{s,U}$)
	$Q_{e,V}$ <————————————	Generate ephemeral key pair ($d_{e,V}, Q_{e,V}$) using (q_e, a_e, b_e, G_e,n_e, and h_e)
Validate $Q'_{e,V}$ using (q_e, a_e, b_e, G_e,n_e, and h_e)		
Compute the shared value ($Z_e = dhp(d_{e,U}, Q'_{e,V})$) using ($q_e$, a_e, b_e, G_e,n_e, and h_e): $Z_e=[h]d_{e,U}Q_{e,V}$		Compute the shared value ($Z_e = dhp(d_{e,V}, Q'_{e,U})$) using ($q_e$, a_e, b_e, G_e,n_e, and h_e): $Z_e=[h]d_{e,V}\,Q'_{e,U}$
Compute the shared value ($Z_s = dhp(d_{s,U}, Q_{s,V})$) using ($q_s$, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,U}Q_{s,V}$		Compute the shared value ($Z_s = dhp(d_{s,V}, Q_{s,U})$)using ($q_s$, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,V}Q_{s,U}$
$Z = Z_e \parallel Z_s$		$Z = Z_e \parallel Z_s$
Determine the shared key: *KeyData = kdf*(*Z*, *keydatalen*, [*SharedData*]		Determine the shared key: *KeyData = kdf*(*Z*, *keydatalen*, [*SharedData*]

5. 7 Full Unified Model with Key Confirmation

This scheme provides an example of how key confirmation could be provided to the Full Unified Model.

Static and ephemeral parameters ((q_s, a_s, b_s, G_s, n_s, and h_s) and (q_e, a_e, b_e, G_e,n_e, and h_e)) are generated for the "system" (a community of users). Each entity generates a static key pair (d_s, Q_s) using the static parameters. The entities must determine the MAC scheme and hash function to be used.

Key Confirmation is not specified in ANSI X9.42. The scheme provides all security attributes except Key Compromise Impersonation resilience to both parties. Interactive communications are required.

Party U	Transmission	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	

Generate ephemeral key pair ($d_{e,U}, Q_{e,U}$) using (q_e, a_e, b_e, G_e,n_e, and h_e)	$Q_{e,U}$ ——————————>	
		Validate $Q'_{e,U}$
	$Q_{s,U}$ – – – – – – – – – – – – – –>	Acquire U's static public key ($Q_{s,U}$)
		Generate ephemeral key pair ($d_{e,V}, Q_{e,V}$)
		Compute the shared value ($Z_e = dhp(d_{e,V}, Q'_{e,U})$) using ($q_e$, a_e, b_e, G_e,n_e, and h_e): $Z_e=[h]d_{e,V}Q_{e,U}$
		Compute the shared value ($Z_s = dhp(d_{s,V}, Q_{s,U})$) using ($q_s$, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,V}Q_{s,U}$
		$Z = Z_e \parallel Z_s$
		Determine the MAC key and the shared key: $MacKey\|\|KeyData = kdf(Z, mackeylen + keydatalen, [SharedData])$
		Create $MacData_1 = 02 \|\| V \|\| U \|\| Q_{e,V} \|\| Q_{e,U} \|\| [Text_1]$
	$Q_{e,V}$, [$Text_1$], $MacTag_1$ <——————————	Create $MacTag_1 = Mac_{MacKey}(MacData_1)$
Validate $Q'_{e,V}$		
Compute the shared value ($Z_e = dhp(d_{e,U}, Q'_{e,V})$)using ($q_e$, a_e, b_e, G_e,n_e, and h_e): $Z_e=[h]d_{e,U}Q_{e,V}$		
Compute the shared value ($Z_s = dhp(d_{s,U}, Q_{s,V})$)using ($q_s$, a_s, b_s, G_s, n_s, and h_s): $Z_s=[h]d_{s,U}Q_{s,V}$		
$Z = Z_e \parallel Z_s$		
Determine the MAC key and the shared key: $MacKey\|\|KeyData = kdf(Z, mackeylen + keydatalen, [SharedData])$		
Create MacData$_1$ = 02 || V || U || $Q'_{e,V}$ || $Q_{e,U}$ || [$Text_1$])		
Verify that the received $MacTag_1$ is correct for		

$MacData_1$		
Create $MacData_2$ = 03 || U || V || $Q_{e,U}$ || $Q'_{e,V}$ || [$Text_2$]		
Create $MacTag_2$ = $Mac_{MacKey}(MacData_2)$	[$Text_2$], $MacTag_2$ ——————————>	
		Create $MacData_2$ = 03 || U || V || $Q'_{e,U}$ || $Q_{e,V}$ || [$Text_2$]
		Verify that the received $MacTag_2$ is correct for $MacData_2$

5.8 Station-to-Station

The system parameters ((q_e, a_e, b_e, G_e, n_e, h_e)) and ((q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig})) are generated for the "system" (a community of users). Each entity generates a signature key pair (d_{sig}, Q_{sig}) using the system parameters (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig}). The entities must agree on the MAC scheme and hash function to be used.

There is no corresponding scheme in ANSI X9.42. This scheme provides all security attributes to both parties. Interactive communications are required.

Party U	Transmission	Party V
Acquire V's signature key ($Q_{sig,V}$)	$Q_{sig,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q_e, a_e, b_e, G_e, n_e, and h_e)	$Q_{e,U}$ ——————————>	
		Validate $Q'_{e,U}$
	$Q_{sig,U}$ – – – – – – – – – – – – – –>	Acquire U's signature key ($Q_{sig,U}$)
		Generate ephemeral key pair ($d_{e,V}$, $Q_{e,V}$) using (q_e, a_e, b_e, G_e, n_e, and h_e)
		Compute the shared value ($Z_e = dhp(d_{e,V}, Q'_{e,U})$) using ($q_e$, a_e, b_e, G_e, n_e, and h_e): Z_e=[h]$d_{e,V}$ $Q_{e,U}$
		Determine the MAC key and the shared key: $MacKey \|\| KeyData = kdf(Z_e, mackeylen+datakeylen,$ [$SharedData$])
		Construct $Data_1$ = $Q_{e,V}$ || $Q'_{e,U}$ || U || [$Text_1$]
		Sign $Data_1$ using $d_{sig,V}$, (q_e,

		a_e, b_e, G_e, n_e, h_e), obtaining $rsig_1$ and $ssig_1$
	$Q_{e,V}$, $rsig_1$, $ssig_1$, [$Text_1$], $MacTag_1$ <————————————	Calculate $MacTag_1 = Mac_{MacKey}(Data_1)$)
Validate $Q'_{e,V}$ using (q_e, a_e, b_e, G_e, n_e, h_e)		
Construct $Data_1 = Q'_{e,V} \| Q_{e,U} \| U \| Text'_1$		
Verify that $rsig'_1$ and $ssig'_1$ are valid for $Data_1$ using $Q_{sig,V}$, (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig})		
Compute the shared value ($Z_e = dhp(d_{e,U}, Q'_{e,V})$) using ($q_e$, a_e, b_e, G_e, n_e, and h_e): Z_e=[h]$d_{e,U}Q_{e,V}$		
Determine the MAC key and the shared key: $Mackey\|KeyData = kdf(Z_e, mackeylen+keydatalen, [SharedData])$		
Verify that the received $MacTag'_1$ is correct for $Data_1$ using 5.7.2		
Construct $Data_2 = Q_{e,U} \| Q'_{e,V} \| V \| [Text_2]$		
Sign $Data_2$ using $d_{sig,U}$, (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig}), obtaining $rsig_1$ and $ssig_2$		
Calculate $MacTag_2 = Mac_{MacKey}(Data_2)$	$rsig_2$, $ssig_2$, [$Text_2$], $MacTag_2$ ————————————>	
		Construct $Data_2 = Q'_{e,U} \| Q_{e,V} \| V \| [Text'_2]$
		Verify that $rsig'_2$ and $ssig'_2$ are valid for $Data_2$ using $Q_{sig,U}$, (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig})
		Verify that the received $MacTag'_2$ is correct for $Data_2$

5.9 One Pass MQV

The system parameters (q, a, b, G, n, and h) are generated for the "system" (a community of users). Each entity generates a static key pair (d_s, Q_s) using the system parameters. The entities must agree on a hash function to be used.

This scheme corresponds to the MQV1 scheme in ANSI X9.42. The scheme provides Implicit Key Authentication to both parties, and Key Compromise Impersonation resilience to the initiator. Interactive or store-and-forward communications may be used.

Party U	Transmission	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q, a, b, G, n, and h)	$Q_{e,U}$ ——————————>	
		Validate $Q'_{e,U}$ using (q, a, b, G, n, and h)
	$Q_{s,U}$ – – – – – – – – – – – – – –>	Acquire U's static public key ($Q_{s,U}$)
Compute the shared value ($Z = mqvp((d_{s,U}, Q_{s,U}), (d_{e,U}, Q_{e,U}), Q_{s,V})$ using (q, a, b, G, n, and h): $implicitsig_U = d_{e,U} + avf(Q_{e,U})\ d_{s,U}$ $R = h\ \acute{}\ implicitsig_U\ \acute{}\ (Q_{s,V} + avf(Q_{s,V})\ \acute{}\ Q_{s,V})$ $Z = x_R$		Compute the shared value ($Z = mqvp((d_{s,V}, Q_{s,V}), Q_{e,U}, Q_{s,U})$ using (q, a, b, G, n, and h): $implicitsig_V = d_{s,V} + avf(Q_{s,V})\ d_{s,V}$ $R = h\ \acute{}\ implicitsig_V\ \acute{}\ (Q_{e,U} + avf(Q_{e,U})\ \acute{}\ Q_{s,U})$ $Z = x_R$
Determine the shared key: KeyData = *kdf*(*Z*, *keydatalen*, [*SharedData*])		Determine the shared key: *KeyData* = *kdf*(*Z*, *keydatalen*, [*SharedData*])

5.10 Full MQV

The system parameters (q, a, b, G, n, and h) are generated for the "system" (a community of users). Each entity generates a static key pair (d_s, Q_s) using the system parameters.

This scheme corresponds to the MQV2 scheme in ANSI X9.42. The scheme provides Implicit Key Agreement, Known-Key Security and Key Compromise Impersonation resilience to both parties, and Forward Secrecy to both parties if explicit authentication is supplied for all session keys. Interactive communications are required.

Party U	Transmission	Party V
Acquire V's static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – –	

Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q, a, b, G, n, and h)	$Q_{e,U}$ ————————————>	
		Validate $Q'_{e,U}$ using (q, a, b, G, n, and h)
	$Q_{s,U}$ – – – – – – – – – – – –>	Acquire U’s static public key ($Q_{s,U}$)
	$Q_{e,V}$ <————————————	Generate ephemeral key pair ($d_{e,V}$, $Q_{e,V}$) using (q, a, b, G, n, and h)
Validate $Q'_{e,V}$		
Compute the shared value ($Z = mqvp((d_{s,U}, Q_{s,U}), (d_{e,U}, Q_{e,U}), Q_{s,V}, Q'_{e,V})$ using (q, a, b, G, n, and h): $implicitsig_U=d_{e,U}+avf(Q_{e,U})$ $d_{s,U}$ $R=h´implicitsig_U´(Q_{e,V}+$ $avf(Q_{e,V})´Q_{s,V})$ $Z=x_R$		Compute the shared value ($Z = mqvp((d_{s,V}, Q_{s,V}), (d_{e,V}, Q_{e,V}), Q_{s,U},Q_{e,U})$ using (q, a, b, G, n, and h): $implicitsig_V=d_{e,V}+avf(Q_{e,V})$ $d_{s,V}$ $R=h´implicitsig_V´(Q_{e,U}+$ $avf(Q_{e,U})´Q_{s,U})$ $Z=x_R$
Determine the shared key: *KeyData* = *kdf*(*Z*, *keydatalen*, [*SharedData*])		Determine the shared key: *KeyData* = *kdf*(*Z*, *keydatalen*, [*SharedData*])

5.11 Full MQV with Key Confirmation

This scheme provides an example of how Key Confirmation could be provided for the Full MQV scheme.

The system parameters (q, a, b, G, n, and h) are generated for the “system” (a community of users). Each entity generates a static key pair (d_s, Q_s) using the system parameters. The entities must determine the MAC scheme and the hash function to be used.

This scheme provides all security attributes to both parties. Interactive communications are required.

Party U	Transmission	Party V
Acquire V’s static public key ($Q_{s,V}$)	$Q_{s,V}$ <– – – – – – – – – – – – – –	
Generate ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using (q, a, b, G, n, and h)	$Q_{e,U}$ ————————————>	
		Validate $Q'_{e,U}$
	$Q_{s,U}$ – – – – – – – – – – – – – –>	Acquire U’s static public key ($Q_{s,U}$)

		Generate ephemeral key pair ($d_{e,V}$, $Q_{e,V}$) using (q, a, b, G, n, and h)
		Compute the shared value ($Z = mqvp((d_{s,V}, Q_{s,V}), (d_{e,V}, Q_{e,V}), Q_{s,U}, Q'_{e,U})$) using ($q$, a, b, G, n, and h): $implicitsig_V = d_{e,V} + avf(Q_{e,V})$ $d_{s,V}$ $R = h´implicitsig_V´(Q_{e,U} + avf(Q_{e,U})´Q_{s,U})$ $Z = x_R$
		Determine the MAC key and the shared key: $MacKey \|\| KeyData = kdf(Z, macdatalen + keydatalen, [SharedData])$
		Create $MacData_1$ = 02 || V || U || $Q_{e,V}$ || $Q'_{e,U}$ || [$Text_1$]
	$Q_{e,V}$, [$Text_1$], $MacTag_1$ <————————————	Create $MacTag_1 = Mac_{MacKey}(MacData_1)$
Validate $Q'_{e,V}$		
Compute the shared value ($Z = mqvp((d_{s,U}, Q_{s,U}), (d_{e,U}, Q_{e,U}), Q_{s,V}, Q'_{e,V})$) using ($q$, a, b, G, n, and h): $implicitsig_U = d_{e,U} + avf(Q_{e,U})$ $d_{s,U}$ $R = h´implicitsig_U´(Q_{e,V} + avf(Q_{e,V})´Q_{s,V})$ $Z = x_R$		
Determine the MAC key and the shared key: $MacKey \|\| KeyData = kdf(Z, mackeylen + keydatalen, [SharedData])$		
Create $MacData_1$ = 02 || B || A || $Q_{e,V}$ || $Q_{e,U}$ || [$Text_1$]		
Verify that the received $MacTag_1$ is correct for $MacData_1$		
Create $MacData_2$ = 03 || U || V || $Q_{e,U}$ || $Q'_{e,V}$ || [$Text_2$]		
Create $MacTag_2 = Mac_{MacKey}(MacData_2)$	[$Text_2$], $MacTag_2$ ————————————>	
		Create $MacData_2$ = 03 || U

		$\|\| V \|\| Q'_{e,U} \|\| Q_{e,V} \|\| [Text_2]$
		Verify that the received $MacTag_2$ is correct for $MacData_2$

6. Key Transport Schemes

No scheme is provided in ANSI X9.42 for key transport.

6.1 One Pass Transport

The system parameters (q, a, b, G, n, and h) are generated for the "system" (a community of users). Each entity who may act as a responder generates a static encryption key pair (d_{enc}, Q_{enc}) using the system parameters and is bound to that key pair (e.g., by a certificate). Each entity who may act as an initiator must be bound to a unique identifier (e.g., in a certificate). The entities must also determine the hash function to be used.

This scheme provides Implicit Key Authentication and Key Compromise Impersonation resilience for the initiator (U) only. Interactive or store-and-forward communications may be used.

Party U	Transmission	Party V
Acquire V's static public encryption key ($Q_{enc,V}$)	$Q_{enc,B}$ <– – – – – – – – – – – – –	
Form the data to be encrypted: $EncData = U \|\| KeyData \|\| [Text]$		
Encrypt *EncData* with V's public key : $EncryptedData = E_{Qenc,V}(EncData, [SharedData_1], [SharedData_2])$	*EncryptedData* ————————————>	
		Decrypt *Encrypted Data* using $d_{enc,V}$ to get *EncData*
		Parse the initiator's identity (U) and *KeyData* from *EncData*
		Verify that *EncryptedData* was received from U

6.2 Three Pass Key Transport

The system encryption and signature parameters ((q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, h_{enc}) *and* (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig})) are generated for the "system" (a community of users). Each entity generates a signature key pair (d_{sig}, Q_{sig}) using the system parameters. Each entity who may act as an initiator generates an encryption key pair (d_{enc}, Q_{enc}) using the system parameters. The entities must also determine the hash function and challenge length to be used.

This scheme provides Implicit Key Authentication, Explicit Key Authentication, Entity Authentication, Key Compromise Impersonation resilience and Unknown Key-share resilience to both parties, and Known-Key Security to both parties when used in conjunction with the elliptic curve augmented encryption scheme. Interactive communications are required.

Party U	**Transmission**	**Party V**
Acquire V's signature and encryption public keys	$Q_{sig,V}$, $Q_{enc,V}$ <– – – – – – – – – – – – – –	
Generate a challenge ($Challenge_U$)	$Challenge_U$ ——————————>	
		Verify that $Challenge_U$ is the correct length
	$Q_{sig,U}$, $Q_{enc,U}$ – – – – – – – – – – – – – –>	Acquire U's signature and encryption public keys
		Generate $Challenge_V$
		Create $EncData = V \parallel KeyData \parallel [Text_1]$
		Encrypt *EncData* and, optionally, $SharedData_1$ and $SharedData_2$ using $Q_{enc,U}$, (q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, h_{enc}): $EncryptedData = E_{Qenc,U}(EncData \parallel [SharedData_1] \parallel [SharedData_2]$
		Create $SignData_1 = Challenge_V \parallel Challenge_U' \parallel U \parallel EncryptedData \parallel [Text_1])$
	$Challenge_V$, *EncryptedData*, $[Text_1]$, $rsig_1$, $ssig_1$ <——————————	Sign $SignData_1$ using $d_{sig,V}$ *and* (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig}) to get $rsig_1$ and $ssig_1$
Verify that the received $Challenge_V$ is the correct length		
Decrypt *EncryptedData* using (q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, h_{enc}) and $d_{enc,U}$ to obtain *EncData*		
Parse V's identity and		

KeyData from *EncData*		
Verify that *EncryptedData* was received from V		
Create $SignData_1$ = $Challenge_V$' || $Challenge_U$ || U || *EncryptedData* || [$Text_1$]		
Verify that the received $rsig_1$'and $ssig_1$'are valid for $SignData_1$ using $Q_{sig,V}$, (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig})		
Create $SignData_2$ = $Challenge_U$ || $Challenge_V$' || V || [$Text_2$]		
Sign $SignData_2$ using $d_{sig,U}$ and (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig}) to get $rsig_2$ and $ssig_2$	$Text_2$, $rsig_2$, $ssig_2$ ——————————>	
		Create $SignData_2$ = $Challenge_U$' || $Challenge_V$ || V || [$Text_2$]
		Verify that the received $rsig_2$' and $ssig_2$' are valid for $SignData_2$ using $Q_{sig,U}$ and (q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, h_{sig})